CS 7638: Artificial Intelligence for Robotics

Solar System (Particle Filter) Project

Fall 2022 - Deadline: Monday October 10th, Midnight AOE

Project Description

The goal of this project is to give you practice implementing a particle filter used to localize a man-made satellite in a solar system. After completing an intergalactic mission, it's time for you to return home. Your satellite is warped through a wormhole and released into your home solar system in perfect circular orbit around the sun. The satellite receives measurements of the magnitude of the collective gravitation pull of the planets in the solar system. Note that this measurement does NOT include the gravitational effects of the sun. Although the gravitational acceleration of other planets can be measured, curiously in your home solar system, the satellite and the planets follow their orbit around the sun and their motion is not affected by other planets.

Your satellite has an on-board model of the solar system that it is being dropped into.

You'll be riding in the satellite which will exit the wormhole somewhere within the solar system, possibly as far off as +/- 4 AU in both X and Y, and it will be ejected into circular counter-clockwise orbit around the sun, which is located at (0,0).

The planets in the solar system are also orbiting counter-clockwise around the sun in circular or elliptical orbits.

You also have at most 300 days to locate yourself and the satellite before food and resources run out.

Note that your software solution is limited to 15 seconds of "real" CPU time, which is different from the simulated "satellite time". Unless your localization and control algorithm is VERY efficient, you will probably not hit the 300-day limit before you run into the CPU timeout. If it takes your localization system more than 100-200 days to determine the satellite location, you may need to improve your localization algorithm.

The gravimeter sensor [sense function] gives you a (noisy) magnitude of the sum across each planet of the gravitational acceleration on the satellite by that planet, +/- some Gaussian noise.

$$v = \left| \sum_{p=1}^{n} \frac{G * M_p}{r_p^2} \right|,$$

where v is the gravity magnitude, n is the number of planets, G is the gravitational constant, M_p is the mass of planet p, and r is the vector from the satellite to the planet.

The body.py file (which you should not modify, but may examine or import) implements the simulated satellite and planets.

The solar_system.py file contains the model for the sun and planets, helper functions to initialize planets in orbit, move them in orbit, or sense the gravimeter measurement from a certain location in the solar system.

The solar_locator.py file contains two functions that you must implement, and is the only file you should submit to GradeScope.

Part A

After warping back to your home solar system, you must localize where you are. The first function is called `estimate_next_pos`, and must determine the next location of the satellite given its gravimeter measurement. If your estimate is less than 0.01 AU from the target satellite's actual (x,y) position, you will succeed and the test case will end.

Note that your function is called once per day (time step), and each time your function is called, you will receive one additional data point. It is likely you will need to integrate the information from multiple calls to this function before you will be able to correctly estimate your satellite's position. The "OTHER" variable is passed into your function and can be used to store data which you would like to have returned back to your function the next time it is called (the next day).

A particle filter may include the following steps and you may want to consider the following questions:

- Initialization
  - How many particles do you need such that some cover the target satellite?
- Importance Weights
  - How does the sigma parameter affect the probability density function of a gaussian distribution and the weights of the particles?
  - Considering the gravimeter measurement has an inverse r-squared relationship to distance, how might using a fixed sigma vs a sigma relative to the gravimeter measurement affect your results?
- Resample
  - How many particles should you keep at each timestep and what are the pros/cons to having more/less particles?
- Fuzz
  - How much positional fuzzing should you have?
  - What percentage of your particles should you fuzz?
- Mimic the motion of the target satellite
- Estimate

Part B

Now that you're back in your home solar system you must send SOS messages back to your home planet, the last planet in the solar system, so that they know to come pick you up. The second function is called `next_angle`. The goal of this function is to set the angle to send a radio message from the satellite to your home planet. Each message you send contains part of your location and trajectory. It takes 10 messages to fully transmit this data. When your home planet receives your messages they will send a team out to retrieve you and your satellite. The test will end once your home planet has received 10 messages.

Your two functions will have the same input (gravimeter), but the next_angle function will return an absolute angle from the satellite to the home planet in radians in addition to the predicted location.

Submitting your Assignment

Please refer to the "Online Grading" section of the Syllabus

Calculating your score

The test cases are randomly generated, and your particle filter is likely to also perform differently on different runs of the system due to the use of random numbers. Our goal is that you are able to generate a particle filter system that is generally able to solve the test cases, not one that is perfect in every situation.

You will receive seven points for each successful test case, even though there are 20 test cases in total (10 in part A, 10 in part B). This means that you only need to successfully complete 15 of the 20 test cases to receive a full score on this assignment. Your maximum score will be capped at 101, although if you are able to solve more than 15 test cases you can brag about it.

Testing Your Code

NOTE: The test cases in this project are subject to change.

We have provided you a sample of 10 test cases where the first two test cases are easier than the actual test cases you will be graded with because they have no measurement noise. These test cases are designed to allow you to test an aspect of the simulation. Test cases 3-10 are more representative of the test cases that will be used to grade your project.

To run the provided test cases on the terminal: `python testing_suite_full.py`

We will grade your code with 10 different "secret" test cases that are similar, but not an exact match to any of the publicly provided test cases. These "secret" test cases are generated using the generate_params_planet.py file that we have provided to you. You are encouraged to make use of this file to generate additional test cases to test your code.

We have provided a testing suite similar to the one we'll be using for grading the project, which you can use to ensure your code is working correctly. These testing suites are NOT complete as given to you, and you will need to develop other test cases to fully validate your code. We encourage you to share your test cases (only) with other students on Ed.

By default, the test suite uses multi-processing to enforce timeouts. Some development tools may not work as expected with multi-processing enabled. In that case, you may disable multi-processing by setting the flag `DEBUGGING_SINGLE_PROCESS` to `True`. Note that this will also disable timeouts, so you may need to stop a test case manually if your filter is not converging.

You should ensure that your code consistently succeeds on each of the given test cases as well as on a wide range of other test cases of your own design, as we will only run your code once per graded test case. For each test case, your code must complete execution within the proscribed time limit (15 seconds) or it will receive no credit. Note that the grading machine is relatively low powered, so you may want to set your local time limit to 3 seconds to ensure that you don't go past the CPU limit on the grading machine.

Academic Integrity

You must write the code for this project alone. While you may make limited usage of outside resources, keep in mind that you must cite any such resources you use in your work (for example, you should use comments to denote a snippet of code obtained from StackOverflow, lecture videos, etc).

You must not use anybody else's code for this project in your work. We will use code-similarity detection software to identify suspicious code, and we will refer any potential incidents to the Office of Student Integrity for investigation. Moreover, you must not post your work on a publicly accessible repository; this could also result in an Honor Code violation [if another student turns in your code]. (Consider using the GT provided Github repository or a repo such as Bitbucket that doesn't default to public sharing.)

Frequently Asked Questions (F.A.Q.)

- *Q* How can I simplify this problem to make thinking about it easier?
  - *A* Take a look at this video that uses a particle filter to solve a 1D problem that has a lot of similarities to this one: Particle Filter explained without equations - https://www.youtube.com/watch?v=aUkBa1zMKv4
- *Q* Are you SURE this can be solved using a particle filter?
  - *A* Yes. See https://mediaspace.gatech.edu/media/t/1_c1f99rxe
- *Q* What is fuzzing?
  - *A* Fuzzing is the process of perturbing (a percentage of) the particles with the intention of diversifying your hypotheses (covering a wider search area). Fuzzing is also known as dithering or roughening (sometimes called jittering). It is discussed in these papers: Sample Impoverishment, PF: Tutorial, Roughening Methods
- *Q* How can I turn on the visualization?
  - *A* Near the top of `testing_suite_full.py` set `PLOT_PARTICLES=True`. Be sure that this is False when you submit to Gradescope. When `PLOT_PARTICLES` is set to True, you will be presented with a visualization that has a cyan square at your estimated (x,y) for the satellite, white dots/triangles representing the particles, and a red triangle showing the target satellite. Planets are lime, and the home planet is magenta with a skyblue trail. While debugging wih visualization, you may consider also setting `DEBUGGING_SINGLE_PROCESS=True`.
- *Q* How can I change the `TIME_LIMIT`?
  - *A* There are additional ALL CAPS variables near the top of the `testing_suite_full.py` file. They are by default set to the same values used by the grader, but you can toggle the True/False values to enable/disable verbose logging and the visualization, and increase the timeout value (logging & visualization slow things down). If your computer is faster than that of Gradescope and iterates through more timesteps, you may consider decreasing the TIME_LIMIT while developing your solution.
- *Q* Why is my local score different from my Gradescope score?
  - *A* The gradescope test cases are different from the student provided test cases in testing_suite_full.py. The gradescope machine may be slower or faster than your machine, causing it to iterate through more or less time steps before timing out. You're encouraged to submit early and often – a failing solution on your local computer might do better on Gradescope and vice versa.

- *Q* Why do I get different results locally when the visualization is on vs off?
  - *A* Since visualization slows things down, with visulization off your solution may get through more timesteps than with visulization on, and those extra timesteps could be where your solution succeeds.